Mansoor Ahmed-Rengers*, Diana A. Vasile*, Daniel Hugenroth*, Alastair R. Beresford, and Ross Anderson

CoverDrop: Blowing the Whistle Through A News App

Abstract: Whistleblowing is hazardous in a world of pervasive surveillance, yet many leading newspapers expect sources to contact them with methods that are either insecure or barely usable. In an attempt to do better, we conducted two workshops with British news organisations and surveyed whistleblowing options and guidelines at major media outlets. We concluded that the soft spot is a system for initial contact and trust establishment between sources and reporters. *CoverDrop* is a two-way, secure system to do this. We support secure messaging within a news app, so that all its other users provide cover traffic, which we channel through a threshold mix instantiated in a Trusted Execution Environment within the news organisation. CoverDrop is designed to resist a powerful global adversary with the ability to issue warrants against infrastructure providers, yet it can easily be integrated into existing infrastructure. We present the results from our workshops, describe CoverDrop's design and demonstrate its security and performance.

Keywords: whistleblowing, anonymous communication, mobile application

DOI 10.2478/popets-2022-0035
Received 2021-08-31; revised 2021-12-15; accepted 2021-12-16.

***Corresponding Author: Mansoor Ahmed-Rengers:** OpenOrigins Limited and University of Cambridge, E-mail: Mansoor.Ahmed@cl.cam.ac.uk
***Corresponding Author: Diana A. Vasile:** Department of Computer Science and Technology, University of Cambridge, E-mail: Diana.Vasile@cl.cam.ac.uk
***Corresponding Author: Daniel Hugenroth:** Department of Computer Science and Technology, University of Cambridge, E-mail: Daniel.Hugenroth@cl.cam.ac.uk
Alastair R. Beresford: Department of Computer Science and Technology, University of Cambridge, E-mail: Alastair.Beresford@cl.cam.ac.uk
Ross Anderson: Department of Computer Science and Technology, University of Cambridge, E-mail: Ross.Anderson@cl.cam.ac.uk

1 Introduction

Since the Snowden leaks [15], newspapers and their potential sources have become aware of the mass-surveillance infrastructure available to nation states. This has profound implications for those who wish to expose wrongdoing within government, and in organisations that can call on its assistance.

Whistleblowers may face severe penalties if caught: in the recent past, they have been physically intimidated [28], imprisoned [46, 51], and even assassinated [45]. Even in less extreme cases, they face dismissal [35], litigation, and professional boycotts [41]. Yet, whistleblowing is often the last line of defence against unethical or illegal behaviour by the powerful. In the context of government agencies, it may be the principal means of exposing crimes that may have been covered up under the guise of national security. It is also widely recognised as a crucial component in accountability, and many countries have explicit laws for protecting whistleblowers. Unfortunately, these laws have proven to be insufficient in many recent cases. Since the Snowden revelations, we have seen the emergence of tools such as SecureDrop, and many major news organisations have web pages to advise whistleblowers on how to contact them securely with sensitive information. A preliminary examination convinced us that the tools are often hard to use securely and the advice inadequate. How could we do better?

In order to understand the whistleblowing process in the real world, we conducted two workshops with journalists and IT staff at news organisations. These workshops provided valuable insights into the practical difficulties of supporting whistleblowers and how journalists work with them. The main lesson we drew from these workshops was that there is no secure way for whistleblowers to initiate contact with reporters. Consequently, initial contact is undertaken via insecure means and only later escalated to more secure channels (e.g., SecureDrop), by which point it may already be too late.

In this paper, we present *CoverDrop*, a system which helps potential whistleblowers securely initiate contact and whose design emerged from these discussions. CoverDrop can be integrated into existing news apps and enables news app users to contact journalists at that news organisation, reducing the risk of insecure communication from the start. It uses cover traffic generated by all the regular users of the news app to hide whistleblowers' communication, where traffic is passed through one or more mixes at the newspaper. Thus, every news app user acts as a potential whistleblower and becomes inconspicuous in the crowd. We fortify this by ensuring that the news app ordinarily sends a small amount of constant cover traffic to the mix (*CoverNode*) hosted in the news organisation's infrastructure; the cover traffic is replaced with message contents when a whistleblower communicates with a reporter.

In its most basic form, a single CoverNode is hosted by the news organisation to protect the privacy of the whistleblower. In order to reduce the attack surface, the CoverNode software operates without storage and runs inside a Trusted Execution Environment (TEE). To provide stronger protection, multiple CoverNodes, run by different news organisations, can be chained together, ensuring that whistleblowers cannot be identified unless *all* the chained CoverNodes are compromised.

CoverDrop provides protection from a global passive network adversary with the ability to issue warrants against infrastructure providers such as the newspaper's network service provider, or even their cloud service provider. It achieves this without disrupting existing content delivery infrastructure: CoverDrop can be integrated with Content Delivery Networks (CDNs) with minimal networking reconfiguration.

In summary, the contributions made in this paper are:

1. The development of a realistic adversarial model and system requirements for an initial contact mechanism based on actual journalists' experiences.
2. A secure mobile library for easy deployment of covert communication channels masked by cover traffic in existing applications.
3. A TEE-based CoverNode that provides strong protection against a powerful adversary and supports 3 million messages per second per CPU core.
4. A networking model that supports integration into existing CDN-based networks with minimal change.
5. Strong protection against a global passive adversary that can also use warrants to compel infrastructure providers and to seize devices from journalists (and their sources).
6. An open-source prototype implementation.

2 Status quo

In order to understand existing systems and workflows, we checked what whistleblowing tools, processes and advice different news organisations offer and analysed their security properties (Table 1), and we organised two workshops with journalists, IT staff and security specialists in British news organisations in late 2019 (§2.2).

2.1 Currently recommended options

We reviewed news organisations' whistleblowing pages. We selected 24 news organisations scoring highly on Alexa top sites, ranking by category and by country. Starting from the home page, we investigated the options they provide for people to contact them (Table 1).

Sadly, we found that only twelve newspapers offer encrypted communication, either in the form of encrypted general-purpose messaging apps or of special-purpose systems such as SecureDrop. We also wanted to know how difficult it is for potential whistleblowers to find this information so, as a heuristic, we counted the number of links a reader has to traverse from the homepage to the relevant whistleblowing page. Of the twelve newspapers offering a secure option, only three link to the information directly from the main page. Most place links two or more hops away. Commendably, the New York Times and The Guardian provide extensive information on the options in an easy-to-understand manner.

Only a few newspaper sites offer PGP keys for specific staff members rather than a company-wide or department-wide account. This observation was later corroborated by journalists in our workshop who stated that having a "front desk" to deal with all leads is standard procedure. This led us to include a generic "any reporter" account as an option in CoverDrop (see §4).

2.2 Workshops

We organised two workshops in London in September 2019 to understand journalists' priorities and procedures (Workshop 1), and capture technical requirements (Workshop 2). We summarise the results below.

We invited journalists and information security experts from large news organisations in the UK (Workshop 1). There were 20 participants (coded as P1 through to P20): 17 in-person, 3 call-ins. Workshop 2 resulted from an invitation to present our approach to one of the news organisations from the first workshop.

Newspaper	Popularity	Easy to find	Postal mail	Telephone	Plaintext email	Online form	Messaging app	Encrypted email	Tor/SecureDrop	Other	Relevant page
New York Times	10m+	2*	●	×	×	×	●	●	●	×	[47]
CNN	10m+	4	×	×	×	×	×	×	×	×	[7]
Times of India	10m+	×	×	×	×	×	×	×	×	×	[31]
BBC	10m+	×	×	●	●	●	●	×	×	●	[27]
The Guardian	5m+	1*	●	●	×	×	●	×	●	●	[16]
Spiegel	5m+	5	●	●	×	●	×	●	●	×	[32]
Le Monde	5m+	5	×	×	×	×	×	×	×	●*	[26]
Washington Post	1m+	×	●	×	●	×	×	×	×	×	[34]
El Pais	1m+	×	●	●	×	×	×	×	×	×	[33]
Süddeutsche Zeitung	1m+	2	●	×	●	×	●	●	●	●	[43]
Wall Street Journal	1m+	×	●	×	●	×	×	×	×	×	[22]
The Mainichi Sh.	1m+	×	×	×	×	●	×	×	×	×	[23]
The Sun	500k+	5	●	●	●	×	●	×	●	×	[42]
China Daily	500k+	×	●	●	●	×	×	×	×	×	[8]
O Globo	100k+	×	×	×	×	●	×	×	×	×	[12]
Buzzfeed	100k+	2	●	●	×	×	●	●	●	×	[29]
Globe and Mail	100k+	1	●	●	●	×	×	●	●	×	[11]
Dawn	100k+	×	×	●	●	×	×	×	×	×	[9]
The Sydney M.H.	50k+	×	×	×	●	×	×	×	×	×	[17]
Wired	10k+	1	×	×	×	×	×	×	●	×	[52]
The intercept	×	1	●	×	×	×	●	●	●	×	[19]
Wikileaks	×	2	×	×	×	×	×	×	●	×	[50]
Private Eye	×	×	●	●	●	×	×	×	×	×	[10]
Ars Technica	×	2	●	×	×	●	●	●	×	●	[4]

Table 1. A view of 24 different newspapers across the globe and the options they provide sources to contact the editorial team. Popularity is calculated by the number of installs from the Google PlayStore. Easy-to-find is calculated as the number of hops from the main web page. *Legend:* ● = *offered*; × = *not offered*; * = *custom solutions offered*

The participants list was refined by the news organisation and we presented to a team of roughly 40 people including journalists, software engineers, and system administrators. We obtained approval from our participants to record the discussion during Workshop 1 to produce a transcript such that the identity and affiliation of the speakers and any other participants is not revealed [18]. The workshop structure and approach were approved by our institution's Ethics Committee.

Each workshop involved presenting the participants our view of the current world (Table 1), as well as an initial presentation of CoverDrop to trigger discussion. We additionally used a semi-structured list of questions to prompt discussion, which covered areas such as: source's contact method, keeping sources anonymous, collaborating with other journalists, and benefits and drawbacks of current systems they use. We conducted a thematic analysis on the transcript from Workshop 1 using double-blind coding to observe recurring themes in the discussion. We then discussed, resolved and refined the themes observed. Although Workshop 2 was more focused on technical requirements due to the mixed audience, we used the opportunity to validate the themes that emerged during the first workshop.

Understanding journalists' priorities. The aim of this workshop was to get a better understanding of how both specialist and mainstream tools support investigative journalism today: what works, what does not, and whether the risks from using these are acceptable.

After the initial presentation it became apparent that no system fits all needs. Bespoke systems for whistleblowing require real understanding of computing, while mainstream applications were fairly easy to use but did not provide the same guarantees. Protection requirements not only differ by context, but also change with time. Although a conversation with a source may initially start off in one channel (which may not have the most careful protection as it's just an initial discussion) it is very common that it *"moves to another channel"* (P4). Also, source cultivation is a long process: *" [sources] want to build up the trust relationship before [giving you the data for the story]"* (P4); and the sources themselves may change in nature: *"[sources] sometimes become whistleblowers because they're getting increasingly angry about something"* (P9). This elevation poses a threat to anonymity since an increase in protection may signal an increase in importance to any listener. Reporters also want a balance between usability and security: *"I think we'd like the easiest solutions that give us with usability the best protections"* (P13).

Whistleblowing is infrequent, with *"less than 10%"*(P4) of user-sent content ever making a story, yet *"you cannot give up the opportunity that some day someone will"* (P13). Securing communication often relies on the journalists since they *"take responsibility [as to what secure tools to use] upon themselves a lot of the time"* (P9). Furthermore, reputation plays an important role in source cultivation: *"the hope is that we can proactively approach an individual with the story based on our reputation for covering that area. Or, alternatively, our reputation is such that people might wish to contact us directly rather than through the shop front."* (P9).

As we explored what a good solution ought to look like, low latency and ease of use became a common factor. Current secure solutions are slow and lack confirmation of message receipt: *"[SecureDrop] is very high latency. [...] someone has to download [messages] on a USB stick and take it to an air-gapped environment and then decrypt it. And even that is quite a slow process."* (P4). Discussion revealed that the real-world latency of SecureDrop is around a day because of all the manual handling. Lower latency (to enable multiple back-and-forth messages in a day) was seen as critical for trust establishment. We worked this into our requirements.

The journalists also repeatedly stated the need for systems that make operational security easy for normal users, that provide resilience to warrants, and that a widespread, trusted whistleblowing system could lead to greater transparency and therefore support democracy.

Technical requirements. The first workshop helped us understand the immediate need: a secure, anonymous and low-latency means for initial contact and trust establishment. One of the organisations that attended the initial workshop invited us to present an early design for CoverDrop. We presented to a team of roughly 40 people, including journalists, software engineers, and system administrators. The presentation was followed by unstructured discussion, which focused on: the importance of blocking the screenshot ability within the app when using CoverDrop; the need for news organisations' app developers to be able to patch the app if vulnerabilities are found; and the use of their organisations' CDNs. Ease of deployment was emphasised in both workshops, with many participants agreeing that the solutions with the highest success rates are those that can be rolled out with minimal engineering effort.

3 Adversarial assumptions

Following our workshops, we summarised the points of failure commonly seen in whistleblowing operations. Based on this and further feedback from journalists, we developed a realistic adversary model.

3.1 Failure points

We focus on the devices involved in the communication between whistleblower and reporter. Failures occur via one of: compromised devices, network adversaries, compromised infrastructure, and operational security mistakes.

Compromised corresponding devices. Assuming the corresponding devices are phones or tablets (either iOS or Android), compromise can occur at several levels: spyware apps, root spyware, temporary adversarial possession, and physical attacks. Root spyware typically allows access to data stored by other apps, as well as UI control and keystroke logging. The latter renders network level protections ineffective. We consider root spyware out of scope for both our system and the existing systems we analyse. (If root spyware is a threat post-disclosure, best practice would be to use CoverDrop only for initial contact, after which the journalist will visit the source to give them a burner phone).

Non-root spyware apps cannot record the screen discreetly due to the warning notification that the OS displays when one application builds an overlay over another. Various side channels have been proposed [6] that allow Android apps to capture partial information about keystrokes entered by other apps, but while such techniques may help guess banking PINs (on easier-to-attack numeric keyboards), their signal-to-noise ratio is too poor for routine police surveillance.

Temporary adversarial possession may occur at border checks or through device confiscation by law enforcement. USB forensic devices can then be used to retrieve system information, assuming the phone has been unlocked since it was powered on. We assume that adversarial possession is possible and only the secure element (SE) and equivalent TEEs could resist such attacks.

Network adversary. We assume that the adversary has access to all of the networking infrastructure (CDN, ISP, corporate LAN); however, we assume that they are generally passive – they will not drop or manipulate messages but rather analyse message traffic to infer connections. For a whistleblower, the knowledge that one has spoken to a reporter at a particular time is incriminating. So they need unobservable communication, not only protecting the message contents but also the fact that any communication occurred at all.

Compromised infrastructure. Warrants allow law enforcement to confiscate material deemed relevant to an investigation. So we must assume that warrants can be issued at any time after investigators learn of the leak. They may be served on third parties such as ISPs, postal services, and cloud providers.

Usability and operational security. Usability is a well-known weakness in security systems [38, 40, 49]; if Johnny can't encrypt, how can we expect him to blow the whistle anonymously? Operational security mistakes can happen at either end of the communication channel and are more likely when journalists are doing something they do rarely, and the source is doing something for the first time in their life under great stress [3].

3.2 Adversary model

Thus, we assume the following adversary model:

1. The adversary can monitor and record all communication to and from the news organisation, its journalists and all potential sources.
2. The adversary can monitor and record all internal traffic within the news organisation.
3. The adversary may issue warrants at any point during or after the leak, allowing them to gain physical control of devices or servers of their choice.
4. The adversary may issue warrants to any third party (CDNs, cloud services, messaging app servers).
5. The adversary can install malicious (non-root) applications on any phone, including those of the reporter and those of all possible sources.
6. The adversary cannot compromise the TEE's confidentiality and remote attestation guarantees during its normal run-time operation. The adversary may issue warrants and perform physical attacks post-disclosure, after which we can assume no guarantees from the TEE.
7. The adversary cannot compromise the mobile phone's secure element (SE), even after taking physical possession of the device.

This adversary is realistic for a senior civil servant who blows the whistle on a government and hopes to remain anonymous. This is the baseline adversary model considered, in § 6.2 we consider stronger adversarial models.

4 CoverDrop overview

4.1 System requirements

We developed functional requirements for CoverDrop during our workshops: *Two way asynchronous communication* to allow for building trust and answering questions. *Low latency* allowing multiple messages to be exchanged per day. *Responsiveness* providing message delivery confirmation. *High usability* in the sense that the UI should be easy for normal people to use and the protocol should not assume any knowledge of cryptography. *Ease of integration* for the news organisation for both the mobile app and networking components.

Whistleblowing involves multiple rounds of communication between the source and the reporters, so excess latency results in further stress for an already anxious whistleblower. We decided to favour low latency over high bandwidth. As such, CoverDrop is not meant for sending large files but rather for text message exchanges to make initial contact and start to establish trust (the journalists repeatedly remarked on the lack of a viable system, §2.2). Thus, CoverDrop can be seen as complementary to systems such as PrivNote and SecureDrop.

4.2 Design

We first give a brief overview of CoverDrop and then dive into the technical details.

Existing systems are inadequate in the face of our adversary model. For example, if one of 67 civil servants must have leaked the document the Prime Minister is fuming about to the Director of the Security Service, then the Director – whose colleagues in the intelligence community operate a global passive adversary – can simply look to see which of them downloaded a Tor client and used SecureDrop (§2). So we decided to take a different approach. Instead of designing a standalone niche application whose use could be a giveaway, we decided to collaborate with news organisations to build CoverDrop into their existing applications.

News organisations have a substantial user base for their apps, which can provide *cover traffic* for covert communications if we introduce constant throughput channels into them. This CoverDrop strategy means that the throughput per user is limited to short text messages (see §4.1 for justification).

SecureDrop was designed using Tor to prevent the news organisation knowing the IP address of the whistleblower. However, this meant that the one civil servant out of 67 who communicated via Tor could expect an 'interview without coffee'. So CoverDrop does not rely on Tor; the source contacts the news organisation directly. We thwart correlation attacks by use of constant-throughput proxies called *CoverNodes* (§6).

Our constant-throughput cover traffic is indistinguishable from the real messages from a source; only by decrypting the feeds can we distinguish the two. In order to ensure that the IP addresses of those who send actual messages aren't accessible to investigators who serve warrants on a service provider, we implement CoverNodes within a Trusted Execution Environment (TEE); in our prototype, this TEE is SGX. We note that one might use multiple CoverNodes across multiple news organisations (as in Tor) for greater resilience to compromise. In this description we focus on the single CoverNode model and highlight changes required for multi-CoverNode in §5.6.

All real messages have two layers of encryption: the outer layer is encrypted with the CoverNode's public key, while the inner layer is encrypted with the recipient's. The data from all the feeds (cover and real) are passed to the SGX enclave where it is decrypted. If the decrypted data indicates a real message, the enclave caches the message until enough messages, cover or real, have arrived and then sends them to the recipients. The enclave never stores any record of messages on disk, so warranted access to the server is ineffective.

In order to defend against spyware on the sending or receiving devices we configure device security with care. We discuss this for Android; similar mechanisms can be employed on iOS. First, we mark our application content as sensitive to ask the operating system to prevent other apps capturing screen content while CoverDrop is in foreground. Second, all app data (active chat logs, etc) are stored encrypted and padded to a fixed size. Every news app user has this encrypted data regardless of whether they have ever used the CoverDrop feature, providing plausible deniability if the device is seized. Third, the master secret for storage encryption is derived from a passphrase and from a secret stored in the secure element (SE). The passphrase gives user control while the SE limits brute-force attacks.

Lastly, in order to meet our ease of integration goal, we designed CoverDrop so that the news organisation's existing CDN infrastructure can route both the cover messages and the real ones.

4.3 Security goals

Our workshops (§2.2) revealed three core security properties required for making initial contact with journalists: confidentiality & integrity, unobservable communication, and infrastructure plausible deniability. We outline CoverDrop's security goals with reference to our adversary model (§3.2) and justify in §6 how CoverDrop achieves these goals.

G1: confidentiality & integrity. The adversary is unable to read or modify the contents of messages between potential sources and journalists.

G2: unobservable communication. The adversary cannot tell which, if any, news app users are currently sending messages to journalists.

G3: plausible deniability. Even with physical possession, the adversary cannot determine which, if any, news app users have previously used CoverDrop.

4.4 Limitations

There are inherent trade-offs between the provision of cover traffic, fixed-size messages, and rate limits. An increase in bandwidth would allow us to support larger or more frequent messages for the minority of users who are whistleblowers, but require the remaining users to send larger volumes of cover traffic. However, our research has determined that there is a sweet spot where the traffic costs for the average user remain low while the bandwidth required to support whistleblowers is sufficient to support first contact with a journalist. We note that messages larger than the predefined size are split into multiple packets and sent over time (due to the rate limits).

We do not consider stronger privacy guarantees, such as perfect forward secrecy and future secrecy, which are becoming the norm for secure messaging apps. Here, they offer little benefit. Perfect forward secrecy, for example, is achievable through changes to the protocol, such as adding session identifiers and key state. However, should key compromise occur, two of CoverDrop's core security goals are defeated since not only does the adversary learn the contents of the message (G1), but they can also now prove the victim was using CoverDrop to communicate with a journalist (G2). Perfect forward secrecy would not recover security past this point.

Lastly, to fully support security goals G2 and G3, CoverDrop relies on the security of TEEs (in our implementation SGX), Secure Element on device, and third-party libraries. We discuss the effect compromise of either of these elements has on the security of CoverDrop in §6.2 and provide an extended analysis of the properties provided by encrypted storage on the mobile device in Appendix B.

5 System details

We first describe the protocol flow before explaining the specific components and actors. As the page limit prevents us explaining all the implementation details, we refer the reader to the source code[1].

We implemented the SGX CoverNode using the Visual Studio SGX SDK in C++. The WebApi is implemented in Python and deployed on a virtual machine behind a well-known CDN provider. The sample NewsReader application and the CoverDrop mobile library are implemented in Kotlin for Android. The mobile library uses libsodium[2] for all cryptographic messages (X25519 key exchange, XSalsa20 stream cipher, Poly1305 as the MAC, and Ed25519 signatures).

5.1 Protocol flow

Our system has four sets of actors: (1) news organisations interested in providing an accessible and trustworthy way for sources to contact reporters; (2) reporters who want to receive newsworthy information and are willing to take measures to protect their sources; (3) whistleblowers who want to share information securely with a reporter while protecting their anonymity; and (4) regular users who simply have the news reader app installed on their devices. They want to minimise any negative implications of participating in the system such as energy consumption and excess mobile data traffic.

We describe the protocol flow as illustrated in Figure 1. The public and private identifiers and keys are

Reporter:	$id_R,\ pub_R,\ priv_R$
CoverNode:	$pub_{SGX},\ priv_{SGX},$ $reporters : \{id_R \rightarrow pub_R\}$
CDN:	$pub_{SGX},\ reporters : \{id_R \rightarrow pub_R\}$
App:	$pub_A,\ priv_A,\ K_{User} \parallel K_{SE}$

Table 2. Private ($priv$) and public (pub) keys of the entities and actors.

summarised in Table 2. We use E to denote private-public-key encryption and S for private-public-key signature. As noted earlier, for simplicity we focus on the single-CoverNode model here; §5.6 summarises the considerations involved in switching to multi-CoverNode.

During the initial setup of CoverDrop all reporters choose a unique, fixed-length identifier id_R and a key-pair $pub_R, priv_R$ for encrypting their messages. Reporters are enrolled by sending a signed tuple (id_R, pub_R) to SGX ⓐ. This sign up process can be modified to interface with any Identity Provider (IDP) that the news organisation might already be using. The SGX enclave (CoverNode) verifies the signature and adds it to its mapping $reporters : \{id_R \rightarrow pub_R\}$. The enclave periodically sends its own pub_{SGX} and the $reporters$ mapping to the WebApi/CDN: `POST /pubkeys` ⓑ.

When the news reader app is first started by any user (other than a reporter) it downloads `/pubkeys` from the CDN ⓒ (or they might come bundled with the app). It creates encrypted storage with a randomly chosen passphrase (see §5.4) and a buffer out of messages to be sent. Every x minutes (where x is a global parameter set by the news organisation) the oldest message from out is sent to the CDN: `POST /message` ⓓ. If out is empty, a "cover" payload is encrypted using pub_{SGX} and sent instead. These cover-traffic messages are detected when being decrypted by SGX and discarded ⓔ.

We now describe the process of starting a CoverDrop session on the app. After navigating to the CoverDrop screen within the app, the user enters a passphrase. CoverDrop then tries to decrypt the existing encrypted storage using $K_{User}+K_{SE}$. If it succeeds, it retrieves its data (including keys and message history) from storage. Otherwise, the existing storage is supplemented with a fresh one (storing pub_A, $priv_A$) and encrypted using fresh $K_{User} + K_{SE}$. It is indistinguishable from any storage that is created during the first app start (see previous paragraph).

The process of sending a message M_A to a reporter is as follows. First, the inner ciphertext to the reporter is created by encrypting the

1 https://github.com/coverdrop/prototype
2 https://libsodium.gitbook.io/doc/

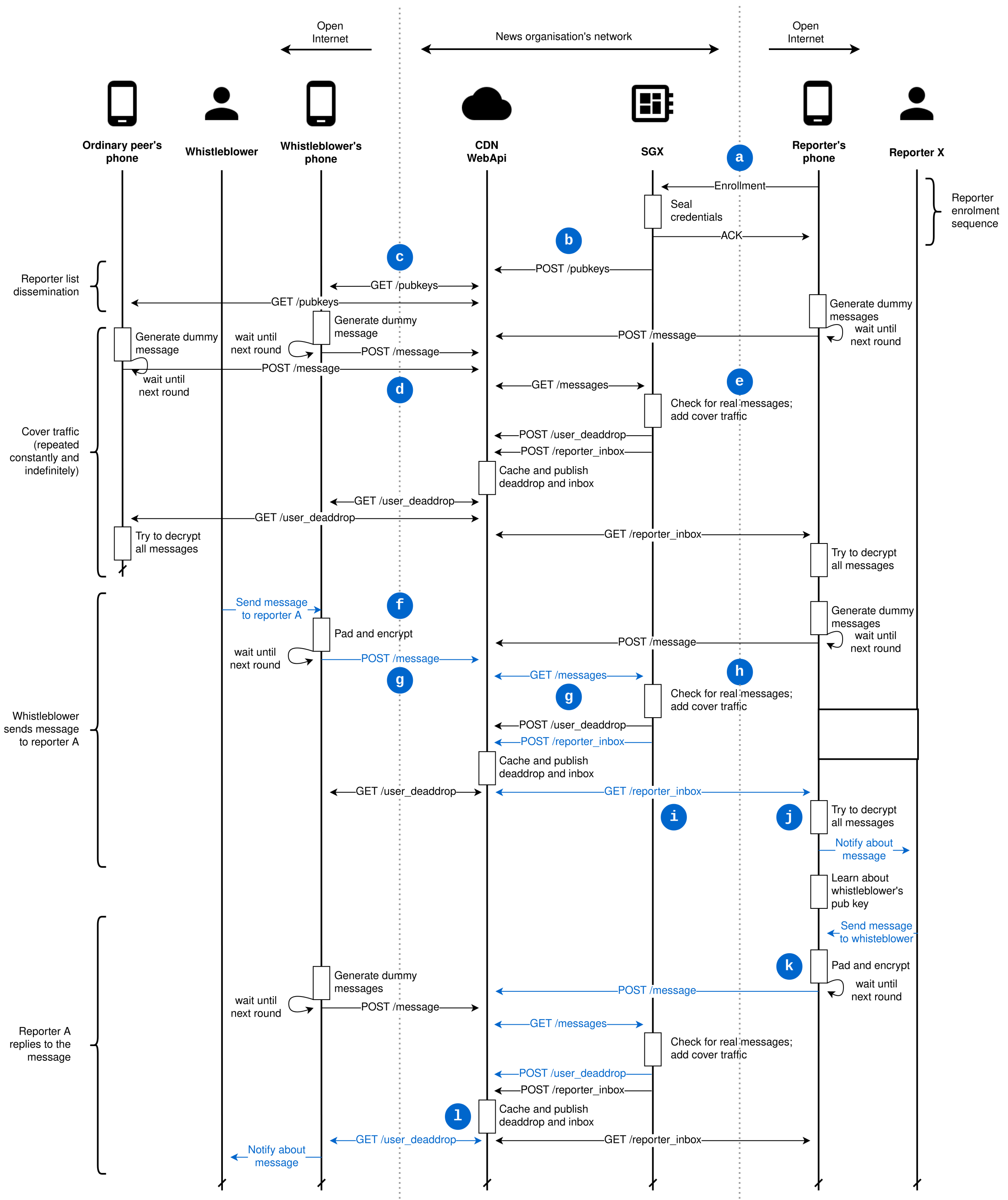


Fig. 1. Protocol flow in CoverDrop.

padded payload and public key ⓕ to the reporter: $C_{A,inner} = E_{pub_R}(pad(M_A, pub_A))$. This inner ciphertext is then encrypted for the CoverNode $C_A = E_{pub_{SGX}}(id_R, C_{A,inner})$ and added to the outgoing buffer *out*.

Second, the message is delivered to the CoverNode via the CDN ⓖ where it will successfully decrypt $C_{A,inner}$. The CoverNode awaits sufficiently many messages and creates cover messages (see §5.2) ⓗ. For all outgoing messages, the CoverNode adds a signature and publishes $(C_{A,inner}, S_{priv_{SGX}}(C_{A,inner}))$ to the reporter inbox.

Third, the reporters download the inbox regularly and verify the signatures ⓘ. They then attempt to decrypt all messages and only succeed for real messages sent to them ⓙ. Now they can read M_A and use pub_A to link the message to an existing chat session (if any).

Fourth, the reporter sends a reply M_R by encrypting it using the sender's public key and adding their signature: $C_{R,inner} = E_{pub_A}(M_R, id_R, S_{priv_R}(M_R))$ ⓚ. They then send $C_R = E_{pub_{SGX}}(C_{R,inner}, S_{priv_R}(C_{R,inner}))$ to the CoverNode. The signature prevents other users from posing as reporters – that would allow them to trick users or fill the dead drop.

Finally, the message is handled by the CoverNode similarly in the other direction. The only difference is that the CoverNode will verify the signature and post it to `user_deaddrop` instead ⓛ. The readers' phones will download the dead drop messages regularly and use trial-and-error to find messages that they can decrypt whenever the user opens their session.

5.2 CoverNode

The SGX enclave acts as a trusted CoverNode hiding the sender/recipient relationships from an observer and providing defence in depth. It is never accessed by the newsreader app directly and never stores any state to disk. Instead, it reads messages from the CDN/WebApi and publishes messages there. This shields it from direct DoS attacks and decouples it from timing attacks.

The CoverNode acts as a Threshold Mix [39] in both directions: forwarding messages from readers to reporters and vice versa. Both work similarly, so we only describe the direction from reader to reporter here.

First, the CoverNode will poll the `/message` endpoint until it has received enough messages to meet its input threshold t_{in} (a configurable parameter). It then decrypts them and learns $(id_R, C_{A,inner})$ for all. All cover messages are discarded while the actual messages $M_{R,actual}$ are collected for each reporter id_R. Second, for each reporter the SGX enclave generates $t_{out} - |M_{R,actual}|$ cover messages where t_{out} is the output threshold. These cover messages are encrypted with a random private key that does not belong to any reporter. Finally, all messages are signed by SGX and published to the reporter inboxes on the CDN/WebApi.

An observer only sees that the CoverNode consumes t_{in} encrypted messages of fixed size and produces t_{out} encrypted messages of fixed size for each reporter inbox. Practically, $t_{in} \gg t_{out}$ as the vast majority of messages are cover traffic.

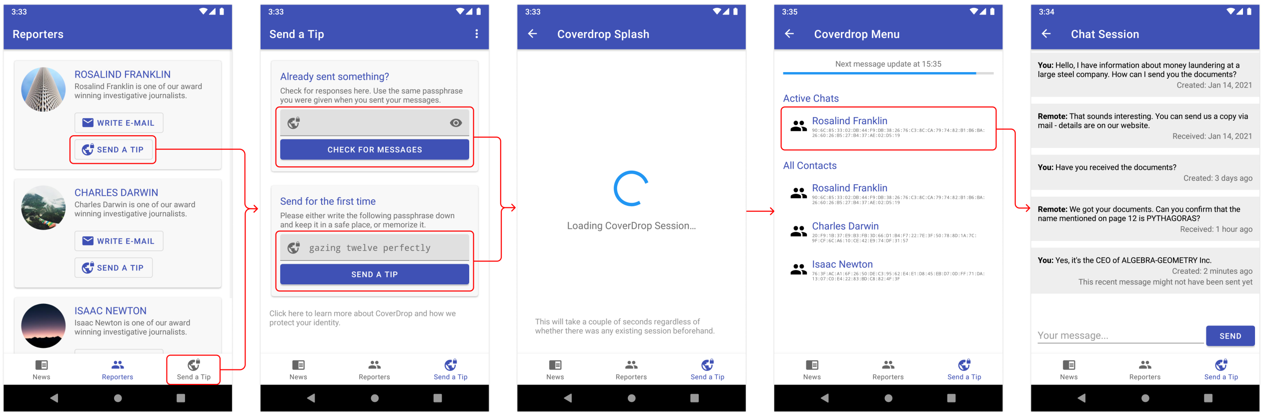
5.3 UI design

The CoverDrop library provides the background services, cryptographic methods, and data interfaces. It does not provide any user interface components itself, allowing native integration with the news organisation's app branding and workflow. We developed a prototype news reader app that shows how it can be integrated.

Our UI design is concerned with ease of access to the CoverDrop features while minimising operational security mistakes. We provide screenshots in Figure 2. The library allows app developers to place entry points of their liking. From our survey of existing channels and workshops we think that a main menu entry and the investigative reporters' profiles are the places where users are likely to expect CoverDrop.

The CoverDrop entry screen always shows options for continuing a previous session and starting a new one. This is because CoverDrop itself only retrieves its state when the encrypted data is loaded with the user's passphrase (if any). This prevents an adversary from knowing if a user has used CoverDrop or not simply by loading the login screen.

The fixed frequency of outgoing messages imposes delays on communication with which the user might be unfamiliar. Thus, we show a progress bar to explain when the next batch of messages will be sent. We impose this fixed frequency to ensure that the whistleblower does not exceed the cover traffic rate and thus become noticeable to a network adversary. In a real application, the integrator might dispense with this, reasoning that no whistleblower making initial contact could reasonably expect a busy journalist to answer within seconds.



ROSALIND FRANKLIN
Rosalind Franklin is one of our award winning investigative journalists.

CHARLES DARWIN
Charles Darwin is one of our award winning investigative journalists.

ISAAC NEWTON
Isaac Newton is one of our award winning investigative journalists.

Already sent something?
Check for responses here. Use the same passphrase you were given when you sent your messages.

Send for the first time
Please either write the following passphrase down and keep it in a safe place, or memorize it.

```
gazing twelve perfectly
```

Click here to learn more about CoverDrop and how we protect your identity.

Loading CoverDrop Session...

This will take a couple of seconds regardless of whether there was any existing session beforehand.

Next message update at 15:35

Rosalind Franklin
```
90:6C:85:33:02:DB:44:F9:DB:38:26:76:C3:8C:CA:79:74:82:B1:B6:BA:
26:60:26:B5:27:B4:37:AE:02:D5:19
```

Rosalind Franklin
```
90:6C:85:33:02:DB:44:F9:DB:38:26:76:C3:8C:CA:79:74:82:B1:B6:BA:
26:60:26:B5:27:B4:37:AE:02:D5:19
```

Charles Darwin
```
20:F9:1B:37:E9:B3:FB:3D:66:D1:B4:F7:22:7E:3F:50:78:8D:1A:7C:
9F:CF:6C:A6:10:CE:42:E9:74:DF:31:57
```

Isaac Newton
```
76:3F:AC:A1:6F:26:50:DE:C3:95:62:E4:E1:D8:45:EB:D7:0D:FF:71:DA:
13:07:C0:E4:22:83:BD:C8:82:4F:3F
```

You: Hello, I have information about money laundering at a large steel company. How can I send you the documents?
Created: Jan 14, 2021

Remote: That sounds interesting. You can send us a copy via mail - details are on our website.
Received: Jan 14, 2021

You: Have you received the documents?
Created: 3 days ago

Remote: We got your documents. Can you confirm that the name mentioned on page 12 is PYTHAGORAS?
Received: 1 hour ago

You: Yes, it's the CEO of ALGEBRA-GEOMETRY Inc.
Created: 2 minutes ago
This recent message might not have been sent yet

Fig. 2. The CoverDrop feature is accessible through reporter profiles and regular navigation. Both login options (new *and* existing session) are always shown. Active sessions and chat logs are only available after decryption of the CoverDrop storage.

5.4 Secure app library

The secure app library allows organisations to integrate CoverDrop into their existing mobile apps. We developed it for recent Android versions and provide a project where we integrate it into a sample newsreader app.

The library registers a background service that runs at fixed intervals and sends encrypted messages to the news organisation. Every epoch, either a real or a cover message is sent. Since we send the same number of fixed-sized encrypted messages at regular intervals, this does not leak any information about whether communication is taking place.

All CoverDrop state is stored in an encrypted file on disk that is padded to a fixed size of 100 KiB. When the app is opened for the first time, the library creates a new encrypted file. This is done using the regular session creation routines with a randomly chosen passphrase pw and a fresh SE key K_{SE}. This ensures that every newsreader installation will contain an encrypted file, providing plausible deniability for whistleblowers. The app updates the last-modified date of the encrypted file on every start-up. During the first start-up the library also chooses a permanent salt value. Our storage implementation uses cryptographic routines provided by the platform and the Argon2 password hash function.

When a user starts a new CoverDrop session with passphrase pw, an empty CoverDrop state $state$ is created and encrypted as follows. First, the empty state $state$ is padded to a fixed size of 100 KiB. Second, a new non-exportable key K_{SE} is created within the SE. Then the padded plaintext is encrypted inside the SE using `AES-GCM` with K_{SE} returning ciphertext $cipher'$. Third, we use a Key Derivation Function (KDF) to derive K_{User} from pw and a salt value. Then the intermediate ciphertext $cipher'$ is encrypted using `AES-CTR` with K_{User} resulting in the final ciphertext.

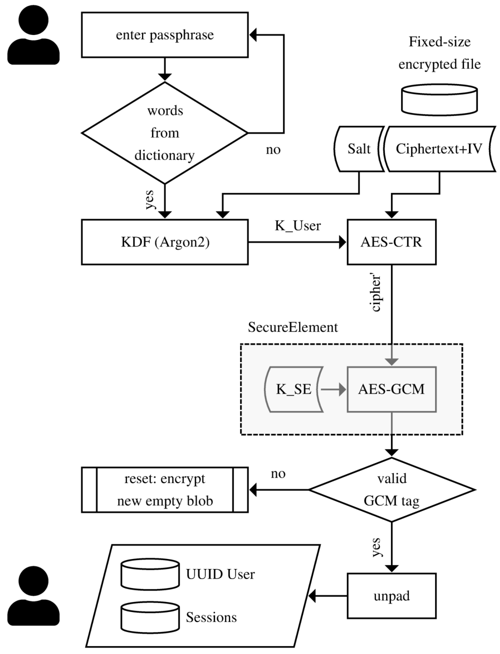


Fig. 3. Flowchart for decrypting the ciphertext stored on disk. The encryption is analogous.

Trying to access/decrypt an existing CoverDrop state is performed analogously (see flow chart in Figure 3). Note that a successful decryption depends on both K_{User} and K_{SE}. Their correctness is not observable until the GCM tag is checked for validity at the last step. Hence an attacker can only verify the passphrases by passing the entire $cipher'$ through the SE.

We assume that the SE resists physical access and does not allow attackers to extract stored keys. Its bandwidth and speed limit the brute-force abilities of an attacker. In our experiments we observed a decryption bandwidth of not more than 20 KiB/s on a Pixel 3 device. This means that an attacker cannot guess more than 12 passphrases per minute. This allows us to generate simpler passphrases from a wordlist that are easy to remember without writing down. When the user enters a passphrase later we check the words against the wordlist to catch common typos. This is important as an incorrect or new passphrase would otherwise indicate that the users wants to start a new session. Our passphrases consist of three words randomly chosen from a 7777 entry list resulting in $\approx 4.70 \cdot 10^{11}$ possible passphrases. Therefore, an attacker is expected to require more than 70,000 years to try all possible combinations.

We discuss this construction and its properties in detail in Appendix B.

5.5 WebApi/CDN integration

The workshops highlighted the need to consider existing network infrastructure and the risk of denial-of-service attacks. Therefore, we decided to allow integration into existing cloud delivery networks (CDNs). CDNs provide distributed caching of static resources and DoS protection through rate-limiting and SSL.

We use distributed caching to disseminate the reporter list and the dead drop containing encrypted messages. These are accessed by all clients regularly. Using the CDN removes this load from the news organisation's web servers. The CDN maintains copies of multiple rounds (e.g. last 7 days) so that clients that missed some rounds (e.g. due to being offline) can catch up.

Where clients send messages (i.e. POST requests) the rate-limiting feature will disallow more POST requests per client than anticipated by the protocol. All POST requests are handled by a web service called *WebApi* instead of going directly to the CoverNode – i.e. it forms a message queue. CoverNodes then collect these from the WebApi on their own schedule, separating them from direct traffic and reducing the risk of exposing the enclave directly to DoS and timing attacks.

5.6 Multiple CoverNodes

As mentioned, CoverDrop can be deployed in a multi-node proxy model i.e., multiple CoverNodes acting in succession. If all the CoverNodes are within the same news organisation, then the changes to the protocol above are straightforward: the sending device picks the desired route for the message, onion-encrypts the message using the public keys of the SGX proxy and sends it to the first hop. At each hop, the SGX enclave decrypts one layer and finds the public key of the next hop, then forwards the packet. The final hop decryption will reveal to the CoverNode whether the message is a cover message or not, where it would be processed as usual.

Things get slightly more complicated if these CoverNodes are located in different organisations. Smaller news organisations may end up being bottlenecks as they struggle to cope with the traffic (cover and real) from larger news organisations. Some form of load balancing, possibly by making the number of SGX nodes proportional to the expected user base, may be needed. A significant benefit of having multi-organisation deployment is that it allows smaller news organisations to benefit from the cover traffic volume of larger peers. An interesting avenue for future work would be to design extensions that allow users of news app A to contact journalists in news organisation B.

6 Security analysis

We first assess whether CoverDrop meets the security goals outlined in §4.3. We discuss in §6.1 each case by reference to an adversary who attempts to defeat each of CoverDrop's security goals and justify why they fail. The adversary only succeeds if they operate outside our threat model; namely, if they are provided with an unlocked CoverDrop session; if a message is waiting in the outgoing message queue *out*; or if the adversary accesses the device before and after the communication without the whistleblower knowing about the first access. Furthermore, our analysis depends on the security of three elements that are used within the CoverDrop infrastructure: TEEs (SGX), on-device SE, and third-party libraries. We discuss in §6.2 what happens to the security offered by CoverDrop when one of these elements is compromised. In Appendix A we also discuss alternative attacks that can lead to the censorship of CoverDrop. Lastly, we discuss the properties of encrypted storage on the mobile device at large in Appendix B.

For the encryption of messages we require an IND-CCA2 secure authenticated encryption scheme and a Diffie-Hellmann key exchange. We implement this in our construction using XSalsa20-Poly1305 and X25519.

6.1 Defeating our security goals

Setup: The Adversary, *Adv*, monitors and records all communication on the public Internet and within the news organisation. *Adv* may also serve warrants to gain physical access to the CDN, CoverNode, and user devices. A whistleblower, *Alice*, wants to contact a journalist, *Bob*, by sending M_A. Adv may attack at the following times: T_a before Alice sends her initial message (Alice only sends regular cover traffic until this time); T_b during the communication (Message M_A is in-flight; Alice has an active CoverDrop session); T_c after the communication has taken place (Alice continues to send cover traffic and has an active CoverDrop session). At any point in time, T_a T_b T_c, we assume there are other app subscribers who opened the app recently, all of which send cover traffic messages.

G1: confidentiality & integrity. CoverDrop provides confidentiality and integrity of messages through authenticated encryption. The sender creates an ephemeral key pair for each message and the encryption layer therefore ensures all ciphertexts are independent.

G2: unobservable communication. At T_a and T_c Alice only sends cover messages and *Adv.* cannot gain any information. At time T_b the message M_A might be part of the traffic that *Adv.* captures between the app and the CoverDrop node. *Adv.* cannot decide if M_A is real or cover since all messages are of the same size and they are all encrypted such that only the CoverNode can decrypt them with its private key. The same is true for messages from the CoverNode to recipients. From this it follows that the the message itself does not leak any information about whether a communication is taking place. Moreover, the timing of the messages is independent of whether any real communication is taking place. Therefore, the scheme achieves unobservable communication.

G3: plausible deniability. We identified three ways through which *Adv* might attempt to defeat the plausible deniability goal, namely whether *Adv* can tell that *Alice* has been using CoverDrop to communicate.

G3.A: *Adv* compromises the CDN. Regardless of the timing of the attack (T_a T_b T_c) the CDN is oblivious to the contents of the messages or final destination of the messages, hence compromising it does not give any advantage.

G3.B: *Adv* confiscates the CoverNode. Before communication T_a, all messages from Alice were for cover, so confiscating the CoverNode gives *Adv* no information. During communication T_b, the CoverNode has knowledge of legitimate messages versus cover messages. However, this information does not leave the SGX enclave and always resides in encrypted memory. Lastly, after delivery of the message T_c, the CoverNode will have deleted all previous messages. This happens after each epoch and if the system is powered down.

G3.C: *Adv* confiscates Alice's device. When confiscating Alice's device at any time, the encrypted storage does not leak information (including resistance against brute-force attempts by the attacker). This is explained in §5.4 and Appendix B. However, during T_b *Adv* could perform three attacks: (i) If there is a message in the outgoing message queue, *Adv* knows that Alice has recently composed a message which has not been sent yet;[3] (ii) If the CoverDrop session is unlocked and in memory, *Adv* can inspect all state; (iii) If *Adv* confiscates Alice's device at two different times, *Adv* can tell if CoverDrop was used in the mean time. This can be countered if Alice is aware of the first access by *Adv* and publicly announces a new (empty) CoverDrop session with an unknown passphrase (e.g. by re-installing the app).

6.2 Compromise of security assumptions

In §6.1 we analyse the security CoverDrop provides in the face of an adversary that challenges our security goals (§4.3). In our analysis we assume that SGX, SE, and third-party libraries are secure. In this section we consider how CoverDrop's security changes if our security assumptions are violated.

SGX compromise. Firstly, we note that we rely on SGX as a defense-in-depth mechanism and still expect the server to be secured in a manner similar to how Tor node operators secure their servers. This means using a dedicated server for the CoverNode and not running unrelated applications on it. Utilising SGX reduces the attack surface available to the adversary but does not eliminate it owing to side-channel attacks discovered on the platform [5, 14, 48]. In case the adversary manages to compromise the CoverNode SGX enclave during runtime in a single CoverNode setup, they can determine which messages are cover and which ones are real. Combined with global network observation, they can also tell

3 This attack can be countered by the app inserting cover messages into the message queue. However, this adds complexity and can lead to out-of-order delivery which has its own usability challenges. Therefore, we leave it for future work.

who send those messages thus successfully executing a correlation attack. The content of the messages remains protected.

In a single-CoverNode environment, some mitigations to reduce the likelihood of compromise include running the latest version of the SGX Platform Software, using newer hardware with mitigations against speculative execution attacks and re-compiling client software with updated SDK [30]. A more robust and effective mitigation would be to move to a multi-CoverNode deployment. In such a deployment, the adversary would have to compromise all the nodes involved in routing the message (not just the entry and exit nodes). Thus, this is an effective mitigation for such an attack especially if the CoverNodes are in different organisations. In settings where competent targeted attacks are likely, newspapers might consider a multi-CoverNode deployment.

Secure Element (SE) compromise. So far, we assumed that the SE of the whistleblower's device resists physical attack. If this assumption is violated (say, new attack methods are discovered), the adversary may brute-force the passphrase that protects the encrypted storage at a much higher rate. If this is a concern (or if the smartphone does not have SE support), the app should generate a more complex passphrase that makes brute-force expensive. Additionally, evidence of communication should be deleted as soon as is practical by creating a new session. This might be achieved by automatically deleting inactive sessions after a set period of time, although this may present usability challenges which we leave to future work.

Third-party libraries. Many mobile applications including news organisations depend on a large number of third-party libraries. Most of them are provided prebuilt and make it very hard for the news organisation to inspect them for malicious behaviour. Others, such as advertisement libraries, intentionally share personally identifiable information (PII) with third parties. Moving towards in-house systems (e.g. NYT's first-party advertisement system [44]) is a positive development not just for source protection but for the privacy of all users.

7 Performance analysis

For our performance discussion, we assume a news organisation and audience with the following properties: There are 1 million users of the news reader application among which there are 100 active whistleblowers. Each whistleblower sends on average 5 real messages per day. (We ignore for now the possible use of the app by ordinary users to give non-sensitive comments and feedback to reporters.) The epoch time is 1h. This means there are about 24 million messages per day from all users and about 21 whistleblower messages per hour. The news organisation has signed up 10 reporters who send on average 10 messages per day.

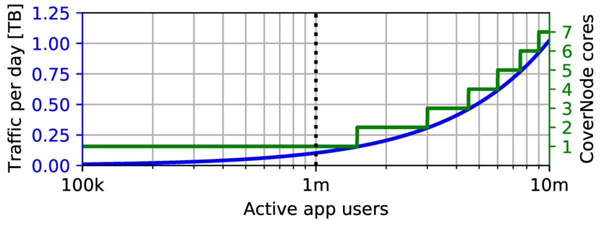


Fig. 4. Total daily traffic for the CDN and required number of CoverNodes cores for different user bases allowing for $2\times$ peak traffic. The dotted line indicates the moderate scenario from our performance discussion. The x-axis uses log-scale.

We now discuss SGX's configuration: For the direction user-to-reporter we let SGX wait for 10^6 messages and output batches (including cover traffic) of 10 messages. Therefore, SGX outputs a total of 240 messages per day to the reporters. In the opposite direction we set the input threshold to 100 messages and the output threshold 10. Therefore, SGX outputs a total of 240 messages per day to the dead-drop endpoint for users. Our results show how traffic and the number of CoverNodes scale linearly for smaller and larger organisations.

7.1 Mobile application

We evaluate our Android prototype using a Google Pixel 3 device running Android 11.

The creation of a cover message requires 7.5ms (σ = 1.4ms) of **CPU time**. With an epoch time of 1h this means less than 200ms in total per day which is small compared to e.g. network operations. Only an actual whistleblower has to decrypt the messages that have been downloaded in the background when they open their CoverDrop session – i.e. the following operations do not apply to the vast majority of users. Each decryption operation takes 7.4ms (σ = 2.4ms). The user will have to perform these for all messages of the user-facing dead-drop (here: 240) and for all reporters from whom they expect a message (here: 1). Thus, they need to perform about 1.8 seconds of work when opening a session every 24 hours. The reporters face a larger amount of messages (here: 2,400), but they only need to check with their own key. Therefore, they also perform about 18 seconds of work per day.

Cryptographic operations increases CPU load and might have an observable thermal effect. We leave the examination of this potential side-channel for future work. Android does not allow access to device-wide CPU usage [13], but thermal information can be read without special permission.

Library size is of concern for mobile applications as they make download times longer and require more disk space. Our prototype library increases the app size by less than 500 KB for most devices (Appendix C). This is small compared to existing news reader applications (usually more than 10 MB in size – Appendix C).

We measured the **data usage** of the network requests of our prototype newsreader app (Appendix D). Sending a (cover) message requires about 6 KB in total including HTTPS overhead - resulting in a total of 4.3 MB per month. However, since the payloads are already encrypted, it could use minimal protocol and achieve less than 0.5 KiB (or 360 KB per month). Downloading the user-facing dead-drop creates 110 KB in traffic per request (or 3.3 MB per month). For comparison: popular news apps use about 1.5 MB when loading the first screen (Appendix D). Individual articles with images typically require downloads larger than 100 KB.

The impact on **Battery Life** is limited as CoverDrop's CPU and data usage is low compared to general news app usage. Thus, the number of messages per reporter and per reader could be increased substantially without degrading device battery life allowing CoverDrop to be used for routine communications between readers and journalists and for confidential tip-offs.

The presented implementation has **API and hardware requirements**. The Hardware Security Element is only accessible from API level 28 (about 40% of all Android devices using the Google PlayStore[4]). On lower API levels the lack of a SE must be mitigated by setting much harder parameters for the password hashing function and not using the wordlist.

CoverDrop provides **easy integration** for app developers. They need to implement the user interface matching their app design, interaction with the synchronous library calls, and four callbacks for interacting with the WebApi via HTTPS. We integrated our library in a prototype news app including all UI functionality in about 900 lines of Kotlin and 1100 lines of XML.

7.2 CoverNode

We implemented the CoverNode in Intel SGX and conducted some performance tests. The CoverNode consists of two components: the trusted enclave where decryption of messages occurs and the untrusted application where network communication occurs. We conducted tests on a laptop with an i7-8565U processor (up to 4.6 GHz) decrypting messages (X25519 and XSalsa20) sent to the CoverNode. With a single threaded enclave, each message takes approximately 1.20 ms when processing in batches of 1000 messages, giving us a throughput of 833 messages per second (3 million messages per hour). A higher batch size reduces processing time since it reduces the number of transitions from untrusted to trusted enclave space which is a relatively expensive operation. It is possible to scale performance by instantiating multiple enclave processes up to the number of cores on the CPU. On our 4-core processor, we achieved linear speed-up with 4 parallel CoverNode processes after which the throughput plateaued. If it is necessary to scale even further, one can use multiple SGX servers in parallel and split the load among them via the CDN/WepApi. As the requests will not be evenly distributed throughout the day, we suggest deploying more CoverNodes than a strict reading of the above numbers would suggest. In Figure 4 we double this number to account for traffic bursts. We have taken care not to use SGX's monotonic counter functionality which has been shown to have limited write cycles [24]. Moreover, no state is stored to disk by the enclave, thus avoiding any concerns about SGX's sealing functionality.

7.3 CDN and WebApi

Most CoverDrop traffic is generated by the news apps downloading the user-facing dead-drop (about 110 KiB, Appendix D) every 24 hours. A user base of 1m results in a total traffic of 102 GB per day (9.95 Mbit/s), which scales linearly with the number of users and configured message throughput. This shows the importance of CDN support in our design, which can very easily distribute the small and static file close to the user. In practice these values are smaller as we can remove users from CoverDrop participation if they have not used the app in the last, say, 7 days. Getting accurate numbers for

4 From the *New Project Wizard* in the Android Studio IDE (version 4.2.2, September 2021).

daily active users (DAU) for news apps is difficult but sources suggest that it may be two orders of magnitude lower than the installed base [21]. This suggests that CoverDrop needs to support less than a million users for the New York Times, which seems on the low side. But as a global network adversary can disambiguate active and inactive app users, our anonymity set is always with respect to the DAU not the number of downloads.

8 Related work

Research into whistleblowing systems is sparse. Volmer et al. propose a submission platform using ad networks to route messages from a source to a relevant news organisation [37], but it cannot guarantee message delivery and it does not withstand our adversary model. McGregor et al. present a grounded-theory approach using in-depth, semi-structured interviews with French and American journalists, showing the need for the computer security community to understand the needs of journalists and study the journalistic process to deal with the significant barriers to communication between journalists and their sources [25]. A Tor-based approach can allow whistleblowers to upload documents to news organisations, but the user requires significant computer security knowledge [20]. Additional hurdles include Tor being banned in many countries and discussions in our workshop noted that using Tor can be a red flag.

Despite limited related research, whistleblowing occurs frequently in practice. We look at the systems commonly in use today and compare these with CoverDrop. Based on the analysis (Table 1) and our workshops (§2.2), we focused on physical mail (*PM*), Whatsapp, Signal, Email, PGP+Email, and SecureDrop.

PM and **email** win on ease-of-use and familiarity. With PM the whistleblower can take precautions without any training and the one-way bandwidth is effectively unlimited; but communication is necessarily one-way (return addresses expose the source), video surveillance makes drop-offs risky, and postage franking leaks approximate geographical location. With email, a source might use a throwaway email address, but warranted access, metadata logging, privacy-intrusive clients, and lack of encryption by default make it very hard to maintain operational security without any computer knowledge, since many email providers' policy require a phone number or recovery email address when signing up.

PGP offers the major benefit that messages are not passed in plaintext, but PGP clients are notoriously difficult for even college-educated people to use [38, 40, 49]. There is also no onion encryption or cover traffic to thwart correlation attacks.

SecureDrop is purpose-built for whistleblowers to share files with reporters anonymously. It uses generated pseudorandom identifiers for the source and does not require any personal data to sign up. It relays all messages over the Tor network, so the news organisation does not know the whistleblower's IP address. This gives it strong resilience to warrants served on ISPs, news organisations and even the whistleblower. We consider SecureDrop to be the closest to CoverDrop in terms of security and infrastructure assumptions (our choice of name itself is homage). However, it has issues. Downloading and using Tor singles out the whistleblower, particularly against a global passive adversary: leaks have shown that the use of privacy-preserving technologies like Tor has put people on NSA watch lists [36]. Tor's plausible deniability set may be small: if only you are using Tor in your office and the leak comes from your office, you may expect trouble. Moreover, installing Tor and navigating to the onion link presents a usability challenge for all but the most tech-savvy whistleblowers. Finally, SecureDrop's latency is a real problem. During our workshops, we learned from reporters that this high round-trip latency (often in the order of days) together with the lack of read receipts has led to potential sources starting a conversation but dropping out due to anxiety.

Signal is fairly popular with $> 10m$ Android installs [1]. Its main weakness lies in its use of mobile phone numbers as identifiers for user accounts. This means that the whistleblower needs to reveal their phone number when communicating with a reporter. This implicates the source despite the end-to-end encryption. Moreover, the Signal server does not implement any network obfuscation, so a global passive adversary might use timing analysis; such an adversary could presumably collect all the traffic to and from the server. Lastly, ISPs can reveal which IP addresses have sent or received Signal messages at a given time, providing another vector of attack for less global adversaries.

WhatsApp is a closed-source messaging app which uses a variant of the Signal messaging protocol. It has a much larger user base than Signal, with more than a billion installs on Android [2]. However, unlike Signal, it collects and stores a substantial amount of metadata about conversations including time of communication, message size, contacts list, etc. As with Signal, WhatsApp uses mobile phone numbers as identifiers.

Other options. Threema differentiates itself from the other systems by not requiring a phone number

	Features	Mail	Whatsapp	Signal	Email	PGP+Email	SecureDrop	CoverDrop
	No personally identifiable IDs	○	×	×	●	●	●	●
Identifiers	System generated random ID	NA	×	×	×	×	●	●
	Encrypts plaintext messages	×	●	●	×	●	●	●
	Resilience to warrants on third parties	×	×	○	×	○	●	●
	Resilience to warrants on news org.	●	●	●	×	○	●	●
Warrants	Resilience to warrants on reporter	●	×	×	×	○	●	●
	Res. to warrant on whistleblower post leak	●	×	×	○	○	●	●
	Res. to warrant on whistleblower during leak	NA	×	×	○	○	●	●
	Resilience to global network adversary	○	×	×	×	×	○	●
Correlation attacks	Effective anonymity set at install	NA	●	○	●	○	×	●
	Effective anonymity set during leak	○	×	×	×	×	●	●
	Resilience to geo. location leakage	×	○	○	○	○	●	●
	Easy-to-use for laypersons	●	●	●	●	×	×	●
	Low latency two-way communication	×	●	●	●	●	×	○
Usability	Verifiable delivery of messages	×	●	●	×	×	×	●
	Spam prevention / rate limits	●	○	○	×	×	×	○
	High throughput	●	●	●	●	●	●	×

Table 3. Whistleblowing systems and features that are fully ●, partially ○, and not × supported.

or email ID for sign-up. However, it requires payment via app stores which raises different privacy concerns. PrivNote is a simple anonymous and encrypted dead-drop site. While this gives good anonymity guarantees it doesn't provide any means of two-way communication, which was considered crucial for trust establishment by journalists at our workshop. It also leaves the question of how to get the note URL to the journalist – an operation where CoverDrop might be an ideal choice.

Finally, in an extreme threat environment, such as when a newspaper has caused real anger to an intelligence agency, it is prudent to expect that root malware will be installed quickly, remotely and covertly on the phones of all relevant suspects and reporters. Such attacks are rare, as the market price for iPhone zero-days is now well into seven figures and such assets are not used casually. However, these exist (as the owner of the Washington Post discovered) and are difficult to block.

The established best practice in high-profile cases of whistleblowing against state actors is for the journalist to meet the source and give them a burner phone, so that they can communicate independently of any devices known to authority. A physical visit also helps the journalist assess the source, not just to establish trust, but to work out whether any other support might be relevant and needful. In such cases, CoverDrop's role is to make contact establishment easier, and much more secure – and thereby to make the news organisations that deploy it the destination of choice for future Snowdens.

9 Conclusion

In this paper we highlighted the shortcomings of the systems currently offered to whistleblowers by the world's major news organisations. We discussed them at length in two workshops with journalists, which enabled us to develop a realistic threat model, and thus a set of requirements for a secure, usable system for trust establishment between sources and reporters. We designed a system for secure initial contact to meet these requirements, called CoverDrop, in consultation with news organisations. CoverDrop uses constant-throughput channels with all users of an organisation's news app acting as sources of cover traffic, giving a large anonymity set in which its sources can hide. Cover traffic and confidential messages are carried over the existing CDN infrastructure to TEE-backed mix nodes hosted by the news organisation. We have demonstrated the security of this scheme against a realistically strong adversary, built a prototype and investigated its performance. We hope to see CoverDrop become the next-generation standard for whistleblower protection across news organisations.

Acknowledgements

We would like to thank our workshop participants and PETS reviewers for their invaluable feedback on this work. Mansoor Ahmed-Rengers was supported by TODAQ and OpenOrigins Limited. Diana A. Vasile and Daniel Hugenroth were partially supported by a Nokia Bell Labs Scholarship, and Daniel also by the Cambridge European Trust. Alastair R. Beresford and Ross Anderson were partially supported by EPSRC [grant number EP/M020320/1].

References

[1] Signal Private Messenger - Apps on Google Play, 2019. https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms.

[2] WhatsApp Messenger - Apps on Google Play, 2019. https://play.google.com/store/apps/details?id=com.whatsapp&hl=en_GB.

[3] Mansoor Ahmed-Rengers, Ilia Shumailov, and Ross Anderson. Snitches Get Stitches: On The Difficulty Of Whistleblowing. In *Proceedings of the 27th International Workshop on Security Protocols*, 2019.

[4] ArsTechnica. Have a confidential news tip for Ars Technica?, 2019. https://arstechnica.com/news-tips/.

[5] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software Grand Exposure: SGX Cache Attacks Are Practical. *arXiv e-prints*, page arXiv:1702.07521, February 2017.

[6] Peng Cheng, Ibrahim Ethem Bagci, Utz Roedig, and Jeff Yan. SonarSnoop: Active Acoustic Side-Channel Attacks, 2018.

[7] CNN. Tips, 2018. http://edition.cnn.com/feedback/tips/.

[8] China Daily. Contact us, 2019. http://www.chinadaily.com.cn/e/static_e/contact.

[9] Dawn. Contact us, 2019. https://www.dawn.com/contact/.

[10] Private Eye. Contact, 2019. https://www.private-eye.co.uk/about/contact.

[11] The Globe and Mail. PGP directory and SecureDrop links, 2018. PGP directory (https://sec.theglobeandmail.com/pgp/) and SecureDrop (https://sec.theglobeandmail.com/securedrop/).

[12] O Globo. Contact us (Portuguese), 2019. https://oglobo.globo.com/fale-conosco/.

[13] Google Issue Tracker. Android o prevents access to /proc/stat, 2017. https://issuetracker.google.com/issues/37140047.

[14] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, New York, NY, USA, 2017. Association for Computing Machinery.

[15] The Guardian. The NSA Files, 2013. https://www.theguardian.com/us-news/the-nsa-files.

[16] The Guardian. How to contact the Guardian securely, 2017. https://www.theguardian.com/help/ng-interactive/2017/mar/17/contact-the-guardian-securely.

[17] The Sydney Morning Herald. Contact us, 2019. https://www.smh.com.au/contact-us.

[18] Chatham House. Chatham house rule. https://www.chathamhouse.org/about-us/chatham-house-rule.

[19] The Intercept. The Intercept welcomes whistleblowers, 2020. https://theintercept.com/source/.

[20] H. Jayakrishnan and R. Murali. A simple and robust end-to-end encryption architecture for anonymous and secure whistleblowing. In *2019 Twelfth International Conference on Contemporary Computing (IC3)*, pages 1–6, 2019.

[21] Joseph Johnson. Daily active users (DAU) of leading iPhone news apps in the United Kingdom (UK) during October 2020, 2020. https://www.statista.com/statistics/878573/leading-iphone-news-apps-dau-united-kingdom/.

[22] Wall Street Journal. Contact us, 2019. https://customercenter.wsj.com/contact.

[23] The Mainichi. Contact form, 2019. https://form.mainichi.jp/mdn/common/content.html.

[24] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1289–1306, Vancouver, BC, August 2017. USENIX Association.

[25] Susan E McGregor, Polina Charters, Tobin Holliday, and Franziska Roesner. Investigating the computer security practices and needs of journalists. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 399–414, 2015.

[26] Le Monde. Contact the editor (French), 2019. https://www.lemonde.fr/faq/?question=28465-contacter-redaction-28465.

[27] BBC News. How to share your questions, stories, pictures and videos with BBC News, 2018. https://www.bbc.co.uk/news/10725415.

[28] BBC News. 'whistleblower' taped to chair and gagged, 2018. https://www.bbc.co.uk/news/uk-scotland-44222575.

[29] BuzzFeed News. Share tips securely & anonymously, 2018. https://contact.buzzfeed.com/?country=en-uk.

[30] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A survey of published attacks on intel sgx, 2020.

[31] The Times of India. Main page, 2019. https://timesofindia.indiatimes.com.

[32] Spiegel Online. How to contact the Spiegel (German), 2019. https://www.spiegel.de/extra/so-nehmen-informanten-sicheren-kontakt-zum-spiegel-auf-a-1030502.html.

[33] El Pais. Contact us (Spanish), 2019. https://elpais.com/estaticos/contacte/.

[34] The Washington Post. Send a letter to the editor, 2019. https://helpcenter.washingtonpost.com/hc/en-us/articles/236004788-Send-a-letter-to-the-editor.

[35] ProPublica. NY Fed Fired Examiner Who Took on Goldman, 2013. https://www.propublica.org/article/ny-fed-fired-examiner-who-took-on-goldman.

[36] Seth Rosenblatt. NSA likely targets anybody who's 'Tor-curious', July 2014. https://www.cnet.com/news/nsa-likely-targets-anybody-whos-tor-curious/.

[37] Volker Roth, Benjamin Güldenring, Eleanor Rieffel, Sven Dietrich, and Lars Ries. A secure submission system for online whistleblowing platforms. In *International Conference on Financial Cryptography and Data Security*, pages 354–361. Springer, 2013.

[38] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent E. Seamons. Why Johnny Still, Still Can't Encrypt: Evaluating the Usability of a Modern PGP Client. *CoRR*, abs/1510.08555, 2015.

[39] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding*, pages 36–52. Springer, 2002.

[40] Steve Sheng, Levi Broderick, Jeremy J Hyland, and Colleen Alison Koranda. Why johnny still can't encrypt: evaluating the usability of email encryption software, 02 2019.

[41] Der Spiegel. Former US Official Reveals Risks Faced by Internal Critics, 2016. http://www.spiegel.de/international/world/ex-us-official-reveals-risks-faced-by-internal-govt-critics-a-1093360-2.html.

[42] The Sun. The sun launches whistleblowers' charter, 2015. https://www.thesun.co.uk/archives/news/142181/the-sun-launches-whistleblowers-charter/.

[43] Süddeutsche Zeitung. So erreichen Sie das Investigativ-Team der Süddeutschen Zeitung, 2020. https://www.sueddeutsche.de/projekte/kontakt/.

[44] The NYT Open Team. To serve better ads, we built our own data program, 2020. https://open.nytimes.com/to-serve-better-ads-we-built-our-own-data-program-c5e039bf247b.

[45] New York Times. Russian Bank Reformer Dies After Shooting, 2006. https://www.nytimes.com/2006/09/15/world/europe/15russia.html?_r=1&oref=slogin.

[46] New York Times. Manning Sentenced to 35 Years for a Pivotal Leak of U.S. Files, 2013. https://www.nytimes.com/2013/08/22/us/manning-sentenced-for-leaking-government-secrets.html.

[47] New York Times. Got a confidential news tip?, 2018. https://www.nytimes.com/tips.

[48] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice. https://sgaxeattack.com/, 2020.

[49] Alma Whitten and J. D. Tygar. Why Johnny Can'T Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, Berkeley, CA, USA, 1999. USENIX Association.

[50] Wikileaks. Submit documents to WikiLeaks, 2016. https://wikileaks.org/Press.html#submit_help_contact.

[51] Wikipedia. Indictment and arrest of Julian Assange, 2019. https://en.wikipedia.org/wiki/Indictment_and_arrest_of_Julian_Assange.

[52] WIRED. How to tip WIRED anonymously, 2019. https://www.wired.com/securedrop/.

A Censorship resilience

An adversary may attempt to censor CoverDrop in two ways: (i) by performing DoS on the CoverNodes or (ii) by disallowing news organisations from using CoverDrop. It is interesting to note that since CoverDrop does not expose any additional servers, all the network connections are routed via existing infrastructure. Thus, unlike with Tor, a nation state cannot simply blocklist IP addresses: they would have to censor the news organisations themselves. The difficulty of doing so depends on the freedom of the press in a given jurisdiction.

To perform DoS on CoverNodes, the adversary would have to either create millions of users (to overwhelm using cover traffic) or send many fake legitimate messages. The first attack vector can be defeated by performing remote attestation with the SE, which would force the adversary to purchase millions of devices; besides (as seen in section §5.2) scaling CoverNodes is inexpensive. The second attack vector is harder to thwart. Since CoverNodes only output a small amount of data in proportion to the input (to reduce overhead for apps), the adversary may be able to force CoverNodes to drop legitimate messages by overfilling their buffers. We can blocklist public keys or devices at the CDN level, although for a nation-state adversary this would only mean a marginal increase in attack cost. This issue is not unique to CoverDrop; other systems with no way to blocklist nodes, such as SecureDrop, can also be overwhelmed by spam. Preventing such attacks remains an open question for whistleblowing systems in general.

B Security properties of the encrypted storage on the mobile device

In this section we argue that the construction for encrypted storage (see §5.4) achieves plausible deniability (including rate limiting password guesses). The construction takes into account the constraints of the Android platform and Secure Element (SE), e.g. regarding available encryption modes. We repeat our construction in pseudocode in Algorithm 1.

We assume that key material cannot be extracted from the SE and that therefore the SE effectively limits the decryption bandwidth. We further assume a stream cipher that is IND-CPA secure – we use AES-CTR in our implementation. We further assume an authenticated encryption scheme that is IND-CPA secure and whose output is indistinguishable from random noise – we use AES-GCM on the SE in our implementation. Finally, we assume that the operating system does not log any file access records other than the last-modified timestamp.

Algorithm 1 Construction for the encrypted storage on the smartphone. The global variable *storage* references the file on disk.

```
1:  procedure OnAppStart
2:      if $storage$ = None then                    ▷ Very first app start
3:          $storage \leftarrow$ AllocateStorageFile()
4:          $storage.salt \leftarrow$ Random()
5:          $storage.iv \leftarrow$ Random()
6:          SE_GenNewKey()                          ▷ Fresh $K_{SE}$
7:          $state \leftarrow$ NewState()
8:          $passphrase \leftarrow$ Random()
9:          Save($passphrase, state$)
10:     Touch($storage$)                            ▷ Updates last-modified
11:
12: procedure Save(passphrase, state)
13:     $K_{user} \leftarrow \mathrm{KDF}_{\mathrm{Argon2}}(storage.salt, passphrase)$
14:     $content_{padded} \leftarrow$ Pad($state$, 100 KiB)
15:     $cipher' \leftarrow$ SE_EncryptAesGcm($content_{padded}$)
16:     $cipher \leftarrow$ AesCtr($K_{user}, storage.iv, cipher'$)
17:     $storage.blob \leftarrow cipher$
18:
19: procedure Load(passphrase)
20:     $cipher \leftarrow storage.blob$
21:     $K_{user} \leftarrow \mathrm{KDF}_{\mathrm{Argon2}}(storage.salt, passphrase)$
22:     $cipher' \leftarrow$ AesCtr($K_{user}, storage.iv, cipher$)
23:     $result \leftarrow$ SE_DecryptAesGcm($cipher'$)
24:     if $result = \perp$ then
25:         return "bad passphrase"                 ▷ GCM
        tag mismatch: the app would then typically ask the
        user if they want to create new storage.
26:     else
27:         return Unpad($result$)
```

Lemma B.1. *The size of the encrypted storage.blob does not leak any information about previous sessions.*

Proof. The construction ensures that each encryption of the current state results in a file of a predefined size. Both active session creation and the initialisation (during the first app start) use the same method Save, resulting in the same size ciphertext. □

Lemma B.2. *The ciphertext cipher in the storage.blob does not leak any information about previous sessions.*

Proof. The inner construction uses an IND-CPA secure encryption scheme (line 15). The second encryption step (line 16) acts as a stream cipher on *cipher′* and does not add information about the state. Therefore the ciphertext *cipher* does not leak information about persistent state. □

Lemma B.3. *The filesystem metadata of the encrypted storage does not leak any information about previous sessions.*

Proof. The fields *storage.iv* and *storage.salt* are only set at the very first start of the news app (lines 4-5) and never updated by CoverDrop sessions.

A recent last-modified date, recorded by the file system, would allow an adversary to identify an active session if it is only updated after CoverDrop use. Our construction updates this information on every start (line 10) and so does not leak this information.

Therefore the filesystem metadata generated by an active session are indistinguishable from those generated by normally use of the app. □

Lemma B.4. *The first decryption step (AES-CTR, line 22) does not leak any information about the key used for decryption.*

Proof. In AES Counter Mode the key is used to initiate a stream cipher that is then XOR-ed with the plaintext (here: *cipher*). The plaintext decrypted by this step is the AES-GCM ciphertext *cipher′* which is itself indistinguishable from random noise. Therefore, the adversary does not gain any information here even if they try the correct key. □

Lemma B.5. *The second decryption step (AES-GCM, line 23) can only be performed on the SE of the targeted device and only at a maximum rate of b/s where b is the throughput of the SE and s is the size of the payload.*

Proof. The key K_{SE} is generated as non-exportable within the SE. The chance of guessing the key in use K_{SE} is prohibitively small (and smaller than the chance of guessing the passphrase). Therefore the decryption cannot be accelerated by using external hardware.

The GCM tag can only be verified after the entire ciphertext (length: $\|cipher'\| = s =$ 100KiB) has been processed by the SE. Therefore, the SE's bandwidth b

effectively limits the overall rate at which an adversary can test keys to b/s. □

Theorem B.6. *The encrypted storage provides plausible deniability and ensures a maximum rate of b/s where b is the throughput of the SE and s is the size of the payload.*

Proof. Lemmas B.1, B.2, and B.3 show that encrypted storage with an active session is indistinguishable from the encrypted storage of a CoverDrop app which has never actively been used. Plausible deniability follows directly from this.

From Lemmas B.4 and B.5 it follows that an adversary has to perform both decryption steps for each password/key they try. This is because the intermediate decryption results do not leak any information. The maximum rate cannot be higher than that of the second decryption step which is b/s by Lemma B.5. □

Theorem B.7. *An adversary that confiscates a device at two points T_1, T_2 in time can determine if CoverDrop was used in between.*

Proof. The adversary can compute a fingerprint of the encrypted storage at T_1 and again at T_2. If CoverDrop is actively used after T_1 and before T_2 the re-encryption of the storage will lead to a new ciphertext if the password or the persisted state changes. □

Note: this information leakage can be be countered by re-installing the app after every device confiscation, or equivalently by announcing a fresh (empty) CoverDrop session with a random password that is not revealed.

C Mobile app size

We analysed the impact on app size by compiling our sample application in release mode using standard optimisation with and without the CoverDrop library. The vast majority of Android devices run either `arm64-v8a` or `armeabi-v7a`. Table 4 shows our results. CoverDrop adds about 160 KB in class files and varying amounts in pre-compiled libraries (.so) such as libsodium for our cryptographic operations. The biggest file in resources is our wordlist file which is about 100 KB uncompressed (about 42 KB compressed).

	arm64-v8a	armeabi-v7a	x86_64
classes.dex	157.9 KB	157.9 KB	157.9 KB
.so files	267.7 KB	340.4 KB	402.2 KB
resources	106.6 KB	106.6 KB	106.6 KB
misc	35.2 KB	35.2 KB	35.2 KB
Final APK	404.7 KB	476.3 KB	537.4 KB

Table 4. Increase of application size for different architectures. The final impact on the distribution file (.apk file) is smaller than the sum of changes due to compression.

This overhead is small compared to the download sizes of the top 7 news apps (all with more than 5m downloads) from Table 1. We summarise the download sizes for Android 11 in Table 5 as provided by the Google PlayStore.

App	Download size
com.nytimes.android	20.49 MB
com.cnn.mobile.android.phone	45.50 MB
com.toi.reader.activities	21.17 MB
bbc.mobile.news.uk	17.14 MB
com.guardian	15.52 MB
de.spiegel.android.app.spon	4.51 MB
com.lemonde.androidapp	14.85 MB
Average	19.88 MB

Table 5. Download size of popular news apps.

D Mobile app traffic

We analysed the traffic generated by the most common requests against the CDN having TLS/SSL enabled and our web service via plain HTTP. The web service is served through an Nginx proxy with GZip compression enabled. The mobile application is using OkHttp3 as its networking library (avoiding connection re-use) and measures the traffic via calls to the Android API `NetworkStats`. We filled the user-facing deaddrop with 240 messages and the public keys endpoint has 3 reporters. The standard payload size is 255 bytes and the user-facing messages have a total size of 385 bytes each, including overhead from the cryptographic operations. Table 6 summarises our findings. The requests via the CDN have a higher overhead due to TLS/SSL.

Operation	RX	TX
HTTP		
GET `/deaddrop`	105.8 KB	4.1 KB
GET `/pubkeys`	0.7 KB	0.4 KB
POST `/user_message`	0.3 KB	1.3 KB
HTTPS via CDN		
GET `/deaddrop`	110.2 KB	3.0 KB
GET `/pubkeys`	4.5 KB	0.8 KB
POST `/user_message`	4.3 KB	1.8 KB

Table 6. Traffic of common requests in our CoverDrop prototype.

As a comparison target, we installed the top 7 applications (all having more than 5m downloads) from Table 1. We started them once to accept the default option of all introduction and popup screens. Then we measured the data usage of the first screen multiple times by first deleting the cache and then starting the app. Table 7 show results of 5 measurements for each app.

App	Traffic
com.nytimes.android	1.11 MB(σ=**0.32** MB)
com.cnn.mobile.android.phone	0.95 MB(σ=**0.24** MB)
com.toi.reader.activities	0.70 MB(σ=**0.10** MB)
bbc.mobile.news.uk	0.43 MB(σ=**0.23** MB)
com.guardian	1.39 MB(σ=**0.33** MB)
de.spiegel.android.app.spon	4.67 MB(σ=**0.20** MB)
com.lemonde.androidapp	1.47 MB(σ=**0.15** MB)
Average	1.58 MB

Table 7. Average data usage for loading the first screen of popular news apps.